A Web-Based IoT Solution for Monitoring Data Using MQTT Protocol

Krešimir Grgić
Faculty of Electrical Engineering, Computer Science and Information Technology Osijek
University of Osijek, Osijek, Croatia
kresimir.grgic@etfos.hr

Ivan Špeh, Ivan Heđi
ICT Department
Virovitica College
Virovitica, Croatia
ivan.speh@vrmti.hr, ivan.hedi@vsmti.hr

***Abstract* —In this paper, a web-based Internet of things solution aimed for monitoring, tracking and analyzing data in agriculture area is proposed. The purpose of the presented solution is to facilitate monitoring of different manufacturing process in mentioned area using IoT technologies. A real time data is achieved using broker-based publishing/subscribing Message Queue Telemetry Transport protocol which is briefly described. Using this protocol limitations resulting from constrained networks in rural areas are avoided. Collected data from sensors is shown and stored in web part of the information system. An architecture of the web application is described as a client-server three-tier architecture in which the graphical user interface (presentation layer), application functions and logic (application layer) and computer data storage (database layer) are developed and maintained as independent modules, on separate platforms. This type of implementation is developed by the manufacturers need for monitoring and tracking data. This architecture is based on future concrete implementation, at least in segments, in several agricultural facilities in Virovitica-Podravina County.**

Keywords—Internet of things; monitoring; MQTT; Raspberry Pi; sensors; three-tier architecture; web application

I. INTRODUCTION

Internet of things (IoT) is no longer future tech trend because of an everyday growing number of objects connected to the Internet worldwide. IoT can be defined as a network of objects which are equipped with sensors and network access modules [1]. IoT object can be any device, vehicle or building from everyday life. With appropriate electronics and software, these objects are able to collect and share data. According to [2] Gartner says that 6.4 Billion connected objects will be in use in 2016., 30% more than in 2015. That number will grow up to 20 billion by 2020. IoT represents the next evolution of the Internet and extends its ability to gather, analyze, and distribute data which can be turned into information and knowledge. Furthermore, IoT has made Internet sensory (temperature, pressure, vibration, light, moisture, stress) thereby achieving improved monitoring, analyzing and tracking systems [3]. IoT is present in almost every human area: society, tourism, education, supply chain management, military, healthcare, industry and many others. This paper is a proposal for IoT implementation in the agriculture area, focused on drying of agricultural products, especially tobacco, chamomile, wheat, fruits or even lumber. Using IoT in mentioned area different improvements are achieved using enhanced tools for real-time monitoring and tracking, decision-making support, automatization of processes, analyzing data and reporting.

II. BACKGROUND

To understand the need for this type of IoT implementation, a short brief of the drying process is explained. Drying process, as the one of the most important postharvest treatment, ensures storage durability, reduces the spoilage and increases the shelf life of agricultural products. To ensure the quality of the final product, the temperature and moisture factors must be monitored during the drying process. If the value of temperature and moisture offset is greater than tolerance setpoint, manufacturers must take additional actions. Although these actions are semi-automated using gas or oil burning stoves, most manufacturers are using wood as fuel. Cost-effectiveness analysis shows that price of drying process using wood as fuel is 70% lower than using oil or gas. Today, almost 80% of small manufacturers are using wood as fuel. One of the main drawbacks of using wood burning stoves is an obligation for temperature surveillance 24 hours a day, seven days a week. Temperature and moisture sensors are already installed in facilities and the values are shown on the control board installed at the front side of the facility. To successfully finish the process, manufacturers must manually check values within a period of the time which is most often a problem, especially during the night. This paper proposes distance monitoring system for temperature/moisture surveillance over the web-based graphical user interface. Already installed sensors are connected to devices (e.g. Arduino Uno, Raspberry Pi) whose main task is to send data (i.e. temperature and moisture values) to a central server over publish/subscribe messaging protocol MQTT. Central server has MQTT broker role, and his main task is to multicast data to subscribed devices. Also, during the real-time monitoring, collected data is sent to web server and SQL server where the data is stored for the purpose of the future analysis and reporting.

III. MESSAGE QUEUING TELEMETRY TRANSPORT

The main characteristic of IoT devices is the machine-2-machine communication which is in this paper presented as a communication between multiple sensor devices and a single data collection device. These sensor devices are most often

978-1-5090-3720-9/16/$31.00 ©2016 IEEE

installed in isolated area where connection to the Internet is limited. For example, most of the facilities are built in rural areas where connection to the Internet is realized over slow DSL connection or cellular network. In such environment, unreliability which reflects high packet loss rate is one of the main disadvantages [4]. There are several protocols proposed for M2M/IoT communication with a focus on mentioned constrained environments. Most frequently adopted protocols are MQTT (Message Queue Telemetry Transport) and CoAP (Constrained Application Protocol). IoT applications can be simplified in the way of the error handling which can be done using these protocols [4].

MQTT is a machine-to-machine (M2M)/Internet of Things connectivity protocol for use on top of the TCP/IP protocol stack which was designed as an extremely lightweight broker-based publish/subscribe messaging protocol for small code footprints (e.g., 8-bit, 256KB ram controllers), low bandwidth and power, high-cost connections and latency, variable availability, and negotiated delivery guarantees [9]. In the hub and spoke model of Message Oriented Middleware messaging server forwards messages from sensor devices to monitor devices [7]. In such architecture, a sensor device whose main task is to continuously produce and send data to server is defined as publisher. Central server, an MQTT broker, collects messages from publishers and examines to whom the message needs to be send. On the other side, every device who had previously registered its interests with a server will keep receiving messages until subscription is canceled (Fig. 1). Using this architecture, publishers and subscribers do not need to know for each other which is one of the major advantages of this protocol. For example, temperature sensor devices need not to know who are the clients that are subscribed for receiving data and conversely [13]. So, the publishers and subscribers do not need to participate in the communication at the same time and do not need to be familiar with each other [16]. It is intended for devices with limited power and memory capabilities, where the network is expensive, has low bandwidth or is unreliable. One of the key requirements of an Internet of Things concept is low network bandwidth used to send data and minimal device resource requirements. While attempting to ensure reliability and delivery, MQTT has met these requirements [12]. This protocol has been applied in a variety of embedded systems. For example, hospitals use this protocol to communicate with pacemakers, oil and gas companies use it to monitor oil pipeline thousands of miles away. Facebook uses this protocol for messaging applications [17]. Fig. 1 shows proposed model of MQTT protocol which runs on TCP/IP connection.

A. MQTT Client

Any IoT object can be MQTT client that sends or receives telemetry data. This could be any device from a microcontroller up to a server. The type of the MQTT client (subscriber or publisher) depends on its role in the system. It can produce or collect telemetry data. In both cases, MQTT client should first connect to a messaging server using a particular type of message explained in the following chapter. After the connection is successfully established, a client must declare himself whether he is a subscriber or publisher. In order to distinguish data sent by the publisher, a topic string is used. For example, a client can publish temperature and humidity values using different string for each value (e.g. temp/25 or hum/40). On the other side, if a MQTT client wants receive the data, he must subscribe to the specific topic. To make a MQTT client from a device, an MQTT library must be installed and connection to an MQTT broker must be established over any kind of network. For example, MQTT client can be a small computer (Arduino or Raspberry Pi) connected to a wireless network with a library strapped to the minimum or a typical computer running a graphical program. The client implementation of the MQTT protocol is simplified and very straightforward [5].

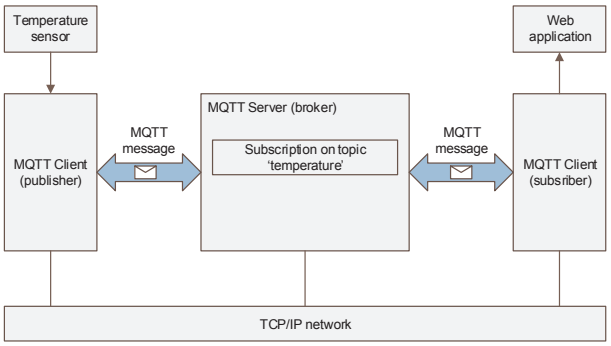

Fig. 1. *Message queuing telemetry transport protocol model*

Client libraries can simplify the process of writing MQTT client applications. MQTT libraries are available for different types of programming languages and platforms, for example, JavaScript, PHP, C, C++, Android, iOS etc. For the initial implementation, Python library will be used.

B. MQTT Broker

An MQTT broker is a central device in mentioned hub and spoke model. The main responsibilities of an MQTT broker are handling communication between MQTT clients and distributing messages between them [4]. A broker can handle up to thousands connected MQTT client at the same time. When the message is received, broker must find all the clients that have a subscription to the received topic [5]. In presented application, it is responsible for receiving messages from the sensors connected to the device with implemented MQTT client library and forwarding it to the designated mobile device. There are plenty other tasks and responsibilities that are handled by the broker. First of all, there is authentication and authorization of the clients for the security purposes. A client can send username and password within connect message that will be checked by the broker. On the broker side topic permission is implemented in order to restrict client to publish or subscribe. Furthermore, for encrypted communication between the broker and the clients, TLS (Transport Layer Security) and SSL (Secure Sockets Layer) encryption is used, the same security protocol that can be used by the HTTP protocol [14]. It is also possible to implement custom authentication or authorization logic into system [5]. There are several message brokers that implement MQTT protocol like Mosquito - open source MQTT v3.1/v3.1.1. broker [18] and HiveMQ – an enterprise MQTT broker [5].

C. MQTT over WebSockets

Using MQTT over WebSockets every browser can be MQTT device. Due the publish/subscribe pattern of MQTT a real-time communication between end device (e.g. temperature sensor) and monitor device (e.g. a web application) is achieved. Using QoS 1/2 message will arrive on the client or broker at least once/exactly once. The broker will queue all messages which client misses when it's not connected. Messages which are retained on the server are delivered when a client subscribes to one of the topics instantly. Paho.js [19] implements this behavior and will be used in initial testing.

D. Message format

The message header for each MQTT command message contains a fixed header with length of 2 bytes and an optional message-specific variable length header and message payload

TABLE I. MQTT MESSAGE

Bit	7	6	5	4	3	2	1	0
Byte 1	Message type				DUP	QoS		RETAIN
Byte 2	Remaining length							

Byte 1 contains the Message type which is represented as a 4-bit unsigned value. There is 14 message type specified in 3.1 version of MQTT protocol. CONNECT (client request to connect to server), PUBLISH (publish message), PUBACK (publish acknowledgment), SUBSCRIBE (client subscribe request) are some of the message types. A detailed description of the message types can be found in protocol specifications [10] and [11]. DUP flag is set when the client the client or server attempts to re-deliver a PUBLISH, PUBREL, SUBSCRIBE or UNSUBSCRIBE message. This applies to messages where the value of QoS is greater than zero (0), and an acknowledgment is required. QoS flag indicates the level of assurance for delivery of a PUBLISH message. The QoS levels are described in next chapter. If RETAIN flag is set to true, normal MQTT message becomes retained message. The broker will restore last retained message with corresponding QoS for specific topic. Each client that subscribes to this topic pattern will receive the retained message immediately. The broker will save only one retained message per topic [5]. Remaining length represents the number of bytes remaining within the current message, including data in the variable header and the payload.

E. Quality of Service

MQTT ensures reliability by providing the option of three QoS levels: At QoS = 0 publisher sends a message at most once and does not check if the message arrived at its destination. No response is sent by the receiver, and no retry is performed by the sender. This lower level is also called "fire and forget" and the message can be lost depending on the network condition [8].

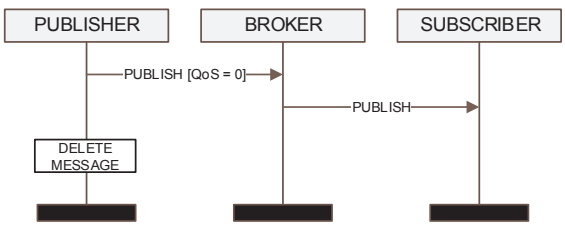

Fig. 2. *At most once delivery* (*MQTT QoS = 0)*

As it is shown in Fig 1., the publisher must send a PUBLISH packet with QoS = 0 and receiver accepts ownership of the message when it receives the PUBLISH packet.

When using QoS level 1, it is guaranteed that a message will be delivered at least once to the receiver. The publisher sends the message and checks the delivery status using PUBACK message. If PUBACK message is lost, the broker will send the same message again, until PUBACK message is received.

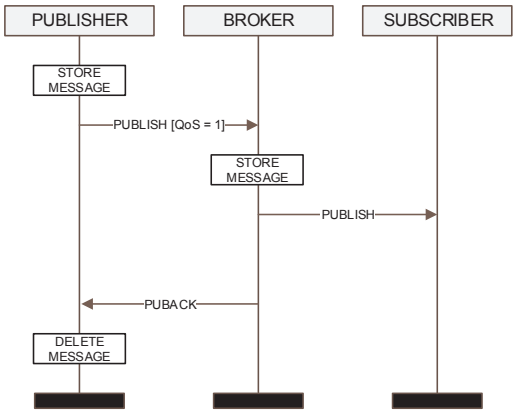

Fig. 3. *At least once delivery* (*MQTT QoS =1)*

In QoS Level 2, also called "assured delivery" and shown in Fig 3., the messages are delivered exactly once using a 4-way handshake. Using QoS level 2 it is possible to have relatively longer end-to-end delay, but there is no messages loss.

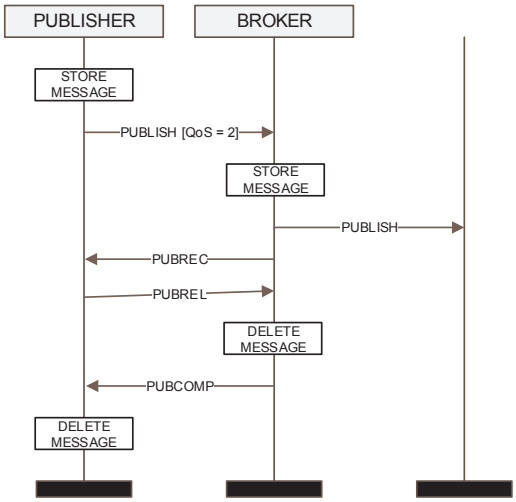

Fig. 4. *Exactly once delivery* (*MQTT QoS =2)*

IV. WEB-PART ARCHITECTURE

An architecture of the web part of the proposed information system can be described as a client-server three-tier architecture in which the graphical user interface (presentation layer), application functions and logic (application layer) and computer data storage (database layer) are developed and maintained as independent modules, on separate platforms.

On the presentation layer, functionality of HTML-based graphical user interface has been extended with custom JavaScript/jQuery functions and several bootstrap friendly jQuery plugins. Using AJAX for communication with Apache web server, features of a single page application are achieved. When a user action generates a client call to retrieve new data, the server only returns view fragments. This means that is possible to update parts of the application, without reloading entire viewport. At the side of design, the bootstrap responsive framework is used. Data received on the MQTT Broker from publishers is automatically forwarded to MQTT client devices where the gauges and diagrams are updated with fresh data (Fig.5.). A central monitor device, also MQTT client, has a task to send data to the Web server in application layer where data is validated and prepared for inserting into SQL tables. There is one central monitor device in system. Saved data can be used to create different types of reports and analyze, which can be shown on other end-user devices in presentation layer. There are two types of ajax communication (both include AJAX request and response) between presentation layer and application layer: saving data (central monitor device) and reading data (all devices in presentation layer). So, data preview on graphical user interface of devices in presentation layer can be directly from MQTT broker, or requested from server with several filters applied (date, time, facility id, process id).

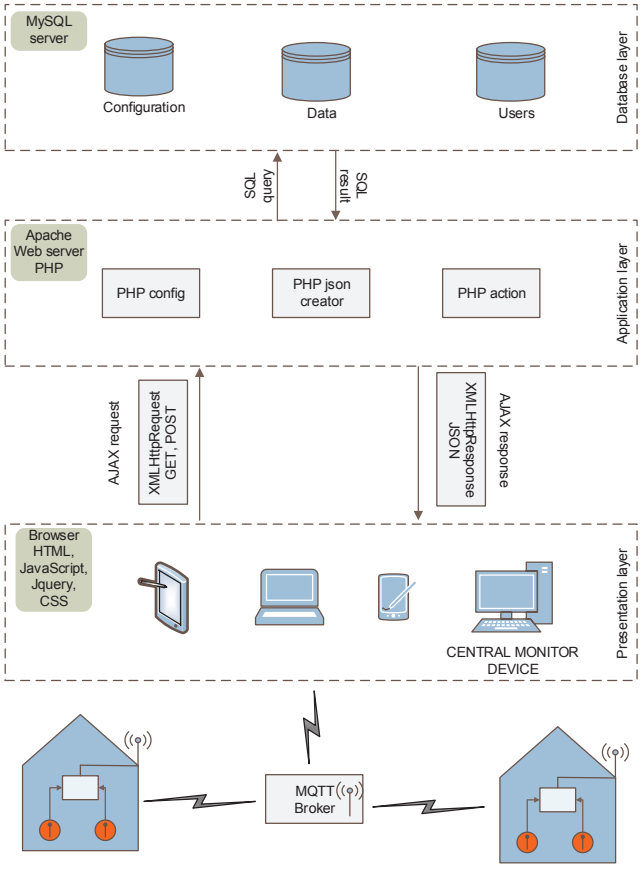

Fig. 5. *Web application architecture*

Ajax request sent by the client is received from the application layer via PHP scripts. There are three types of PHP scripts:

1) PHP script that handles user's actions and receives data from MQTT broker – used for inserting, updating and deleting data from data server
2) A PHP script that prepares data – after a request for data is received, this type of script is used for connecting and retrieving data from MySQL server which is then encoded in JSON format and sent back to client
3) configuration PHP – used for manipulating with session storage, database connection, and other application configuration variables.

On the database layer, there are several MySQL database servers where data is stored and retrieved. Data in this tier is kept independent of application servers. There is the database used for configuration storage, data storage and domain with user storage.

V. Implementation Details

A prototype of proposed system is implemented in a tobacco drying kiln type SD – 78/2 (7270mm x 3150mm x 2850mm) with two chambers (Fig. 6) and integrated wood burning stove TD – 80 (Fig. 7).

Fig. 6. *Drying kiln with inside chambers*

On the front side of the facility, the control board is attached to the kiln. On the control board, there are several switches used for managing and monitoring drying process. As mentioned, for initial test RaspberryPi model 3 is used. Main differences between RPi model 3 and prior versions are integrated wireless module (802.11n), integrated Bluetooth module (Bluetooth 4.1) and 1.2GHz 64-bit quad-core ARMv8 CPU. Using this type of RPi there is no need for additional modules for connecting end device to the network. Using 1602A LCD screen attached to the control board, temperature and humidity values are shown on the board.

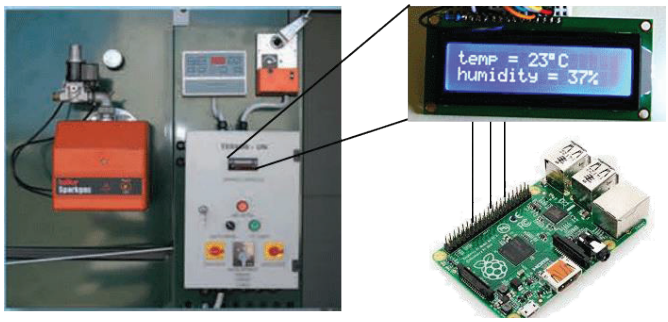

Fig. 7. *Raspberry Pi with connected display attached to burning stove*

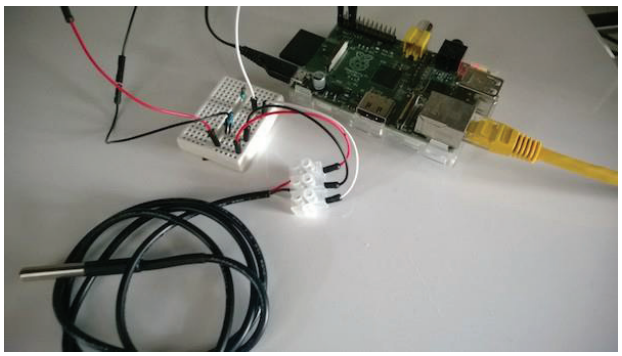

Fig. 8. *Connected temperature sensor in lab environment*

Temperature measurement is achieved using the DS18B20 sensor that provides 9-bit to 12-bit Celsius temperature measurement. DS18B20 communicates with RaspberryPi over one data line [20]. After connecting the sensor to RPi as shown in Fig. 8, data can be sent to the central server (broker). For reading data from sensors and sending values to the broker, Python language with Paho Python MQTT library is used. In this library, a client class is implemented and RPi is extended with MQTT support [5].

```
import paho.mqtt.client as paho
import time
client = paho.Client()
client.on_publish = on_publish
client.connect("193.198.57.183", 1883)
client.loop_start()
 while True:
    temperature = read_temperature()
    (rc, mid) = client.publish
("test/temp", str(temperature), qos=1)
    time.sleep(30)
```

Fig. 9. *Publishing data with Python*

The portion of code in Fig. 9 shows how to publish data. After the client is created, it connects to the broker and starts the background network thread. It then starts a loop reading from sensors every 30 seconds and publishing the value to the topic *test/temperature.* On the other side, an open source MQTT Mosquito broker installed on linux server is used to multicast data to subscribed devices. After receiving data on the subscribed device, data is sent from central monitor device to WEB and SQL server as shown in the previous chapter. For the purpose of central monitor device, PC is used.

VI. Conclusion

This paper has proposed IoT solution for realizing real-time web-based solution aimed for monitoring and tracking temperature and moisture values in the agricultural drying process. The focus was on reliable delivery of data from sensors to end user devices using broker-based publish/subscribe messaging protocol MQTT, which main characteristics are low overhead, asynchronous communication, low complexity and low power. Using this protocol limitations resulting from constrained networks in rural areas are avoided. In proposed model the client producing data (publisher) sends data to the server which forwards data to end user devices. Also, the architecture of web-based application aimed at showing results collected from sensors is presented. Architecture is described as client-server three-tier architecture with three separate layers – presentation, application and database layer.

References

[1] F. Xia, L.T. Yang, L. Wang, and A. Vinel, "Internet of things," International Journal of Communication Systems, vol. 25, pp. 1101-1102, 2012.

[2] S. Charmonman, P. Mongkhonvanit, V. Dieu, and N. Linden "Applications of internet of things in e-learning," International Journal of the Computer, the Internet and Management, vol.23, no.3, pp. 1-4, September-December 2015.

[3] D. Evans, "The internet of things – how the next evolution of the internet is changing everything," white paper, April 2011.

[4] L. Dürkop, B. Czybik, and J. Jasperneite, "Performance evaluation of M2M protocols over cellular networks in a lab environment," Intelligence in Next Generation Networks (ICIN), February 2015, pp. 70-75.

[5] HiveMQ Enterprise MQTT Broker. (2016, Feb. 8). [Online]. Available: http://www.hivemq.com/.

[6] Scalagent, "JoramMQ, a distributed MQTT broker for the internet of things," white paper, September 2014.

[7] Scalagent, "Benchmark of MQTT servers," white paper, January 2015.

[8] E.P. Frigieri, D. Mazzer, and L. Parreira, "M2M protocols for constrained environments in the context of IoT: A comparison of approaches," in International Telecommunications Symposium.

[9] V. Gazis et al., "A survey of technologies for the internet of things," in International Wireless Communications and Mobile Computing Conference, August 2015, pp. 1090-1095.

[10] International Business Machines Corporation (IBM), "MQTT V3.1 Protocol Specification," August 2010.

[11] A. Banks, R. Gupta, "MQTT Version 3.1.1," Octobar 2014.

[12] V.Lampkin et al., Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry, First Edition, September 2012.

[13] D. Thangavel, X. Ma, A. Valera, H. Tan, and C.K. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," Intelligent Sensors, Sensor Networks and Information Processing, April 2014.

[14] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," Transaction on IoT and Cloud Computing, vol 1., January 2015.

[15] K. Tang et al., "Design and implementation of push notification system based on the MQTT protocol," in International Conference on Information Science and Computer Applications, September 2013.

[16] S.K. Shriramoju, J. Madiraju, and A.R. Babu, "An approach towards publish/subscribe system for wireless networks," International Journal of Computer and Electronics Research, vol. 2., pp. 505-508, August 2013.

[17] E.G. Davis, A. Calaveras, and I. Demirkol, "Improving packet delivery performance of publish/subscribe protocols in wireless sensor networks," Sensors, 2013.

[18] Eclipse Mosquitto. (2016, Feb. 12). [Online]. Available: http://mosquitto.org/-

[19] Eclipse Paho JavaScript Client. (2016, Feb. 12). [Online]. Available: https://eclipse.org/paho/clients/js/.

[20] Maxim Integrated. (2016, Feb. 14). [Online]. Available: www.maximintegrated.com.